Compaq Portable Mathematics Library

Order Number: AA-PV6VE-TE

April 2001

This document provides reference and exception information for CPML, the Compaq Portable Mathematics Library software.

Revision/Update Information: This manual supersedes Compaq Portable Mathematics Library, Version 7.1

Software Version: OpenVMS Alpha Version 7.3
OpenVMS VAX Version 7.3

Compaq Computer Corporation
Houston, Texas

ZK6118

The Compaq OpenVMS documentation set is available on CD-ROM.

This document was prepared using DECdocument, Version 3.3-1b.

Contents

Glossary

Index

Tables

Preface

The Compaq Portable Mathematics Library (CPML) is a set of mathematical routines that are accessed from high-level languages (such as Fortran and C) which support mathematical functions. Many CPML routines can also be called directly using standard call interfaces, but it is recommended that you invoke CPML routines only from a high-level language.

Intended Audience

This book is for compiler writers, system programmers, and application programmers who want to use CPML routines.

Document Structure

This manual consists of the following:

Chapter 1 gives a general overview of the mathematics library and discusses supported data types, exception behavior, and IEEE considerations.

Chapter 2 explains the presentation format of a CPML routine and how to interpret a routine's interface. It also alphabetically lists and describes the routines.

Appendix A lists the floating-point boundary values used by the CPML routines.

Appendix B contains the complete list of entry-point names.

The Glossary lists mathematical terms and symbolic names used in this manual, and provides a brief definition.

Related Documents

Some books in Compaq's documentation sets help meet the needs of several audiences. For example, the information in some system books is also used by programmers. Keep this in mind when searching for information on specific topics.

Use the documentation overview and the master index information for your operating system when searching for hardcopy information on a topic. They provide information on all of the books in your operating system's documentation set.

CPML Documentation

For additional information about CPML, you can access the Compaq CPML website at the following location:

`http://www.compaq.com/math`

OpenVMS Documentation

For additional information about Compaq OpenVMS products and services, access the Compaq website at the following location:

`http://www.openvms.compaq.com/`

How to Order Additional Documentation

Use the following World Wide Web address to order additional documentation:

`http://www.openvms.compaq.com/`

If you need help deciding which documentation best meets your needs, call 800-282-6672.

Reader's Comments

Compaq welcomes your comments about this manual. Please send comments to one of the following:

Internet	**openvmsdoc@compaq.com**
Mail	Compaq Computer Corporation OSSG Documentation Group, ZKO3-4/U08 110 Spit Brook Rd. Nashua, NH 03062-2698

Conventions

In this book, every use of OpenVMS means Compaq's OpenVMS operating system, and every use of UNIX means Compaq's Tru64 UNIX operating system.

The following conventions are used in this manual:

Ctrl/x	A sequence such as Ctrl/x indicates that you must hold down the key labeled Ctrl while you press another key or a pointing device button.
PF1 x	A sequence such as PF1 x indicates that you must first press and release the key labeled PF1 and then press and release another key or a pointing device button.
Return	In examples, a key name enclosed in a box indicates that you press a key on the keyboard. (In text, a key name is not enclosed in a box.) In the HTML version of this document, this convention appears in brackets, rather than a box.
. . .	A horizontal ellipsis in examples indicates one of the following possibilities: • Additional optional arguments in a statement have been omitted. • The preceding item or items can be repeated one or more times. • Additional parameters, values, or other information can be entered.
. . .	A vertical ellipsis indicates the omission of items from a code example or command format; the items are omitted because they are not important to the topic being discussed.

( )	In command format descriptions, parentheses indicate that you must enclose choices in parentheses if you specify more than one.
[ ]	In command format descriptions, brackets indicate optional choices. You can choose one or more items or no items. Do not type the brackets on the command line. However, you must include the brackets in the syntax for OpenVMS directory specifications and for a substring specification in an assignment statement.
|	In command format descriptions, vertical bars separate choices within brackets or braces. Within brackets, the choices are optional; within braces, at least one choice is required. Do not type the vertical bars on the command line.
{ }	In command format descriptions, braces indicate required choices; you must choose at least one of the items listed. Do not type the braces on the command line.
bold text	This typeface represents the introduction of a new term. It also represents the name of an argument, an attribute, or a reason.
italic text	Italic text indicates important information, complete titles of manuals, or variables. Variables include information that varies in system output (Internal error *number*), in command lines (/PRODUCER=*name*), and in command parameters in text (where *dd* represents the predefined code for the device type).
UPPERCASE TEXT	Uppercase text indicates a command, the name of a routine, the name of a file, or the abbreviation for a system privilege.
`Monospace text`	Monospace type indicates code examples and interactive screen displays. In the C programming language, monospace type in text identifies the following elements: keywords, the names of independently compiled external functions and files, syntax summaries, and references to variables or identifiers introduced in an example.
-	A hyphen at the end of a command format description, command line, or code line indicates that the command or statement continues on the following line.
numbers	All numbers in text are assumed to be decimal unless otherwise noted. Nondecimal radixes—binary, octal, or hexadecimal—are explicitly indicated.

1

Introduction to CPML

The Compaq Portable Mathematics Library (referred to as CPML) includes a wide variety of mathematical routines that cover the following areas:

- Floating-point trigonometric function evaluation
- Exponentiation, logarithmic, power function evaluation
- Hyperbolic function evaluation
- Algebraic function evaluation
- Complex function evaluation
- Complex exponentiation
- Miscellaneous function evaluation

This manual documents the CPML routines and, in particular, how they behave when given an exceptional input argument. It also documents operating system entry points and supported floating-point data types.

1.1 Overview

Developing software within the confines of high-level languages like Fortran and C greatly increases the portability and maintainability of your source code. Many high-level languages support mathematical function evaluation. CPML was developed to provide a common set of routines that supports many of the common mathematical functions across a wide variety of operating systems, hardware architectures, and languages.

In most cases, the common mathematical functions behave in the same way for all languages and platforms. Occasionally, however, high-level language definitions of the same mathematical function differ for specific input values. For example, in Fortran, log(-1.0) causes a program abort, while in C, log(-1.0) quietly returns a system-defined value.

This document uses the term **exceptional arguments** to refer to values in the following situations:

- Values for which high-level languages disagree on the function behavior
- Values that are mathematically undefined or out of range
- Values for which the function would overflow or underflow

See Section 1.3 for more detail on exceptional arguments.

To provide uniform quality of mathematical functions for all languages on your system, CPML traps exceptional arguments and invokes a system-specific routine called the CPML exception handler. The exception handler is designed to work with high-level language compilers and run-time libraries (RTLs) to provide specific language semantics for exceptional arguments. This means that the

user-visible behavior of a given function called from a given language is not necessarily determined by the routines in the CPML library but rather by a combination of several entities acting in concert.

Note

Compaq strongly recommends that you limit your access to the CPML routines documented in this manual to the high-level language syntax of your choice, thereby guaranteeing the behavior of the routines across platforms. Because of the complex relationship between high-level languages and CPML routines, the behavior of direct calls to CPML routines may change from release to release.

1.2 Data Types

CPML is designed to support mathematics function evaluation for multiple data types. These data types include integer, floating-point, and complex floating-point.

The integer data type, identified as int throughout this manual, is the natural size signed integer for a particular platform. On a 32-bit system, int is a 32-bit signed integer, and on a 64-bit system, int is a 64-bit signed integer.

The floating-point types referred to in this document are F_FLOAT, G_FLOAT, X_FLOAT, S_FLOAT, and T_FLOAT, respectively. When it is not necessary to distinguish between the different floating types, they are referred to collectively as F_TYPE. Your platform may support all or a subset of these floating-point data types. For example, CPML on OpenVMS Alpha systems supports the following floating-point data types: VAX single- and double-precision, IEEE single- and double-precision, and IEEE extended-precision. CPML on Compaq Tru64 UNIX Alpha systems supports only IEEE single- and double-precision data types. Table 1–1 describes the floating-point data types.

Table 1–1 Floating-Point Data Types

F_TYPE	Description
S_FLOAT	32-bit IEEE single-precision number
T_FLOAT	64-bit IEEE double-precision number
X_FLOAT	128-bit IEEE extended-precision number
F_FLOAT	32-bit VAX single-precision number
G_FLOAT	64-bit VAX double-precision number

In addition to the data types mentioned in Table 1–1, CPML also provides routines that return two values of the same floating-point type, for example, two S_TYPE values or two G_TYPE values. In the discussion that follows, these pairs of floating-point data type values are referred to as F_COMPLEX. Refer to Table 1–2. This document uses F_COMPLEX to indicate that a given routine returns two different values of the same floating-point data type.

The mechanism for returning two floating-point values from CPML routines varies from platform to platform. However, on OpenVMS Alpha systems, F_COMPLEX data is returned in consecutive floating-point registers and is accessible only through a high-level language, like Fortran, that specifically allows access to it.

A complex number, z, is defined as an ordered pair of real numbers. The convention used in this manual to define an ordered pair of real numbers as complex is as follows:

- The first number is the real part of the complex number.
- The second number is preceded by i and is the imaginary part of the complex number.
- A separator character (plus sign) is used to associate and separate the real and the imaginary number.

For example:

$z = x + iy$
$z = \sin x + i\cos y$

CPML includes complex functions, for example, the complex sine, csin(x,y), defined to be sin(x + iy). Complex function routines like csin( ), which have complex input, accept floating-point numbers in pairs and treat them as if they are real and imaginary parts of a complex number.

In the previous two examples, the first floating-point values are defined by x and sin x, respectively, and are the real part of the complex number. The second floating-point values used in the examples are defined by iy and icos y, respectively, and are the imaginary part of the complex number. Similarly, CPML routines that return complex function values return two floating-point values. Taken together, these two floating-point values represent a complex number.

CPML supports the floating-point complex types described in Table 1–2. You can access CPML complex functions only through high-level languages that support the complex data type. Use only the data types supported by your system.

Table 1–2 Floating-Point Complex Data Types

F_COMPLEX	Description[1]
S_FLOAT_COMPLEX	An ordered pair of S_FLOAT quantities, representing a single-precision complex number
T_FLOAT_COMPLEX	An ordered pair of T_FLOAT quantities, representing a double-precision complex number
X_FLOAT_COMPLEX	An ordered pair of X_FLOAT quantities, representing an extended-precision complex number
F_FLOAT_COMPLEX	An ordered pair of F_FLOAT quantities, representing a single-precision complex number
G_FLOAT_COMPLEX	An ordered pair of G_FLOAT quantities, representing a double-precision complex number

[1]The lower addressed quantity is the real part; the higher addressed quantity is the imaginary part.

1.3 Exceptional Arguments

Not all mathematical functions are capable of returning a meaningful result for all input argument values. Any argument value passed to a CPML routine that does not return a meaningful result, or is defined differently for different environments, is referred to as an exceptional argument. Exceptional arguments that result in an exception behavior are documented in the Exceptions section of each CPML routine in Chapter 2.

Exceptional arguments typically fall into one of two categories:

- Domain errors or invalid arguments. These are arguments for which a function is not defined. For example, the inverse sine function, asin, is defined only for arguments between -1 and +1 inclusive. Attempting to evaluate acos(-2) or acos(3) results in a domain error or invalid argument error.
- Range errors. These errors occur when a mathematically valid argument results in a function value that exceeds the range of representable values for the floating-point data type. Appendix A gives the approximate minimum and maximum values representable for each floating-point data type.

1.4 Exception Conditions and Exception Behavior

CPML routines are designed to provide predictable and platform-consistent exception conditions and behavior. When an exception is triggered in a CPML routine, two pieces of information can be generated and made available to the calling program for exception handling:

- A notification that an exception has occurred. The mechanics of exception notification vary from platform to platform (for example, signaling, trapping, set errno).
- A return value. If your environment allows your routine to continue after raising an exception condition (with an exception handler for example), then a return value is made available upon completion of the routine.

The exception condition-handling mechanisms on your platform dictate how you can recover from an exception condition, and whether you can expect to receive an exception notification, a return value, or both, from a CPML routine.

The Exceptions section of each CPML routine documents each exceptional argument that results in an exception behavior. In addition to the exceptional arguments, an indication of how the CPML routines treat each argument is given. Exceptional arguments are sometimes presented in terms of symbolic constants.

For example, the following table lists the exceptional arguments of the exponential routine, exp(x):

Exceptional Argument	Exception Condition/Routine Behavior
x > ln(max_float)	Overflow
x < ln(min_float)	Underflow

The exceptional arguments indicate that whenever x > ln(max_float) or x < ln(min_float), CPML recognizes an overflow or underflow condition, respectively.

The symbolic constants ln(max_float) and ln(min_float) represent the natural log of the maximum and minimum representable values of the floating-point data type in question. The actual values of ln(max_float) and ln(min_float) are described in Appendix A.

CPML recognizes three predefined conditions: overflow, underflow, and invalid argument. Table 1–3 describes the default action and return value of each condition.

Table 1–3 Default Action and Return Values for Exception Conditions

Exception Condition	Default Action	Return Value
Overflow	Trap	HUGE_RESULT
Underflow	Continue Quietly	0
Invalid argument	Trap	INV_RESULT

The values HUGE_RESULT and INV_RESULT are data-type dependent.

For IEEE data types, HUGE_RESULT and INV_RESULT are the floating-point encodings for Infinity and NaN, respectively.

For VAX data types, HUGE_RESULT and INV_RESULT are max_float and 0, respectively.

1.5 IEEE Std 754 Considerations

The Institute of Electrical and Electronics Engineers (IEEE) ANSI/IEEE Std 754-1985, IEEE Standard for Binary Floating-Point Arithmetic data types include denormalized numbers (very close to zero). The standard supports the concept of "Not-a-Number" or NaN to represent indeterminate quantities, and uses plus infinity or minus infinity (so that they behave in arithmetic) like the mathematical infinities. Whenever a CPML routine produces an overflow or indeterminate condition, it generates an infinity or NaN value.

All CPML routines, except one, return a NaN result when presented with a NaN input. The only exception is pow(NaN,0) = 1 in ANSI C.

1.6 X/Open Portability Guide Considerations

Table 1–4 lists the routines described in this manual that conform to the requirements of the X/Open Portability Guide, Version 4 (XPG4), or are implemented as UNIX extensions to the XPG4 standard (XPG4-UNIX). Descriptions of these routines appear in Chapter 2 under the generic function name listed in Table 1–4. Platform-specific entry-points are listed in Appendix B.

Table 1–4 XPG4 Conformant Routines

Routine	Conforms to Standard	Generic Function Name
acos	XPG4	acos
acosh	XPG4-UNIX	acosh
asin	XPG4	asin
asinh	XPG4-UNIX	asinh
atan	XPG4	atan
atan2	XPG4	atan
atanh	XPG4-UNIX	atanh
ceil	XPG4	ceil
cos	XPG4	cos

(continued on next page)

Table 1–4 (Cont.) XPG4 Conformant Routines

Routine	Conforms to Standard	Generic Function Name
cosh	XPG4	cosh
cot	XPG4	cot
erf	XPG4	erf
erfc	XPG4	erf
exp	XPG4	exp
expm1	XPG4-UNIX	exp
fabs	XPG4	fabs
floor	XPG4	floor
fmod	XPG4	fmod
frexp	XPG4	frexp
gamma	XPG4	lgamma
hypot	XPG4	hypot
ilogb	XPG4-UNIX	ilogb
isnan	XPG4	isnan
j0	XPG4	bessel
j1	XPG4	bessel
jn	XPG4	bessel
ldexp	XPG4	ldexp
lgamma	XPG4	lgamma
log	XPG4	log
log10	XPG4	log
log1p	XPG4-UNIX	log
logb	XPG4-UNIX	logb
modf	XPG4	modf
nextafter	XPG4-UNIX	nextafter
pow	XPG4	pow
remainder	XPG4-UNIX	remainder
rint	XPG4-UNIX	rint
scalb	XPG4-UNIX	scalb
sin	XPG4	sin
sinh	XPG4	sinh
tan	XPG4	tan
tanh	XPG4	tanh
y0	XPG4	bessel
y1	XPG4	bessel
yn	XPG4	bessel

2 CPML Routines

CPML routines can be accessed from high-level languages that support mathematical functions (such as Fortran and C), or called directly using standard call interfaces. It is highly recommended that you invoke CPML routines only from a high-level language.

CPML routines are documented with generic names, and with the symbol F_TYPE to indicate generic floating-point values (e.g. F_ TYPE sqrt (F_TYPE x)).

To determine the appropriate names and interfaces within a specific programming language (e.g. float sqrtf(float x) or REAL*4 SQRT), refer to that language's documentation.

To enable the use of CPML routines which are not provided by your high-level language, the actual CPML entrynames are provided.

Note: CPML routines which return complex numbers ("F_COMPLEX") use a private interface. Therefore, they can only be called from high-level languages that support that interface.

The Data Types S_FLOAT, T_FLOAT and X_FLOAT refer to IEEE format floating-point numbers of single-, double-, and quad-precision, respectively. F_FLOAT and G_FLOAT refer to VAX format single- precision, and G-floating double-precision floating point numbers, respectively.

For each CPML routine, "exceptional" input values are also provided. That is, values for which the function is mathematically undefined, or for which the output would be out of range for the floating-point type.

Refer to your language's documentation for information about how exceptions manifest themselves and how to control exception behavior.

Further information is also available at the Compaq Math website at: http://www.compaq.com/math.

2.1 CPML Routine Descriptions

CPML routines are described in detail at the end of this chapter. Each CPML routine documented in this chapter is presented in the following format:

- Routine name—A brief name to identify the function of the routine. A routine may contain more than one function.
- Interface—What the routine expects to receive and what it returns. See Section 2.2 for more information.
- Description—Additional information, including the permitted range of input values and generic calculations used to compute the results.
- Exceptions—A description of how the routine behaves when given a specific exceptional input argument.

2.2 CPML Routine Interface

The interface to each function is:

RETURN_TYPE generic_interface_name (INPUT_ARG_TYPE...)

Each of these is described below.

RETURN_TYPE
The data type of the value that the routine returns to your application program. Each routine returns a specific class of data type. For example, either F_TYPE or F_COMPLEX can appear in a CPML interface as described in Chapter 2. The supported data types are described in Section 1.2.

generic_interface_name
The generic name. CPML routines in this chapter are listed in alphabetic order by their interface names. Some CPML routines may be available in the syntax of your high-level language. Fortran and C are examples. To maximize the portability of your application, use the corresponding mathematical routine described in your high-level language, and directly call only the routines documented in this manual that are not supported by your language. Refer to Appendix B for the specific entry-point names needed to directly call a CPML routine from your platform.

INPUT_ARG_TYPE...
The number and type of input arguments provided by your application. Some routines require more than one argument. Arguments must be coded in the order shown in the interface section of each routine described in this chapter. The supported data types for arguments are described in Section 1.2.

Note

Unless otherwise noted, arguments are read-only and passed by value. Arguments passed by another mechanism are prefaced by an asterisk (*); for example, *n in the frexp( ) routine.

2.3 Specific Entry-Point Names

Each generic interface name documented in the interface section of a routine description corresponds to one or more specific entry-point names described in Appendix B. For example, on OpenVMS Alpha systems, the acosd function has five entry-point names, one for each available floating-point data type. The acosd entry-point names are math$acosd_f, math$acosd_s, math$acosd_x, math$acosd_g, and math$acosd_t. On Compaq Tru64 UNIX Alpha systems, the acosd function has two entry-point names corresponding to their supported data types: S_FLOAT and T_FLOAT. The two entry-point names are acosdf for S_FLOAT input arguments and acosd for T_FLOAT arguments. Use the specific entry-point name that corresponds to the input argument data type.

2.4 Working with Exception Conditions

Each CPML routine description contains a table of exceptions. Each exception listed in the table represents an exceptional case that is handled in a platform-specific manner. For example, the atan2( ) exception table contains the following two entries:

Exceptional Argument	Routine Behavior
y = x = 0	Invalid argument
|y| = |x| = infinity	Invalid argument

The first entry describes an exception condition containing two input arguments with zero values. Upon detecting this error, the routine behavior signals the "invalid argument" condition. The second entry is applicable only to platforms supporting signed or unsigned infinity values. Here, if the absolute value of both input arguments is equal to infinity, an "invalid argument" condition is signaled.

The exact behavior of a routine that detects an exceptional argument varies from platform to platform and is sometimes dependent on the environment in which it is called. The behavior you see depends on the platform and language used. It also depends on how the routine was called and the interaction of the various layers of software through which the call to the routine was made. Remember, access to a CPML routine can be made either through direct access (a CALL statement written by a programmer in a source code statement) or through indirect access (from compiler-implemented mathematical syntax).

The default behavior for detecting the x=y=0 arguments is to generate an exception trap when accessing atan2( ) indirectly through Fortran compiler syntax. C compiler syntax for the atan2( ) routine sets errno and returns a NaN when give the same input. In these cases, your compiler documentation provides you with information on how to work with exception conditions.

2.5 CPML Routine Interface Examples

This section discusses the atan2( ) and cdiv( ) interfaces and explains how to interpret them. The explanations given in this section apply to all CPML routines.

2.5.1 atan2( ) Interface

The interface to the atan2( ) routine is:

F_TYPE atan2 (F_TYPE y, F_TYPE x)

The routine name atan2( ) is the high-level language source-level name that gets mapped to a specific entry-point name documented in Appendix B. This is the name that appears in compiler documentation for this mathematical routine. The appropriate entry-point name is automatically selected when atan2( ) is called from high-level language syntax. This selection depends upon the data type of the input arguments. If you make direct calls to this routine, you must manually select the proper entry-point name documented in Appendix B for the data type of your input arguments.

The format of the atan2( ) routine shows that it expects to receive two input arguments by value. Both arguments must be the same F_TYPE. The returned value will also be the same F_TYPE as the input arguments.

For example, on OpenVMS Alpha systems, the G_FLOAT entry-point name is math$atan2_g( ). It takes two G_FLOAT arguments by value and returns a G_FLOAT result.

For Compaq Tru64 UNIX Alpha systems, the S_FLOAT entry-point name is atan2f( ). The routine takes two S_FLOAT input arguments by value and returns an S_FLOAT result.

2.5.2 cdiv( ) Interface

The interface to the cdiv( ) routine is:

F_COMPLEX cdiv (F_TYPE a, F_TYPE b, F_TYPE c, F_TYPE d)

The routine name cdiv( ) is the generic name that gets mapped to a specific entry-point name documented in Appendix B. Selection of the appropriate entry-point name is done automatically when cdiv( ) is called from high-level language syntax. This selection depends upon the data type of the input arguments. Again, if you make direct calls to this routine, you must manually select the proper entry-point name documented in Appendix B for the data type of your input arguments.

The format of the cdiv( ) routine shows that it expects to receive four input arguments by value. All arguments must be the same F_TYPE. The returned value will be an F_COMPLEX data type and will be the same base data type as the input arguments.

For example, on OpenVMS Alpha systems, the F_FLOAT entry-point name is math$cdiv_f( ). This routine takes four F_FLOAT input arguments by value and returns an F_FLOAT_COMPLEX result in an ordered pair of F_FLOAT quantities.

For Compaq Tru64 UNIX Alpha systems, the S_FLOAT entry-point name is cdivf( ). This routine takes four S_FLOAT input arguments by value and returns an S_FLOAT_COMPLEX result.

acos - Arc Cosine of Angle

Interface

F_TYPE acos (F_TYPE x)

F_TYPE acosd (F_TYPE x)

Description

acos( ) computes the principal value of the arc cosine of x in the interval [0,pi] radians for x in the interval [-1,1].

acosd( ) computes the principal value of the arc cosine of x in the interval [0,180] degrees for x in the interval [-1,1].

Exceptions

Exceptional Argument	Routine Behavior
$|x| > 1$	Invalid argument

acosh - Hyperbolic Arc Cosine of Angle

Interface

F_TYPE acosh (F_TYPE x)

Description

acosh( ) returns the hyperbolic arc cosine of x for x in the interval [1,+infinity]. acosh(x) = ln(x + sqrt(x**2 - 1)).

acosh( ) is the inverse function of cosh( ). The definition of the acosh( ) function is acosh(cosh (x)) = x.

Exceptions

Exceptional Argument	Routine Behavior
x<1	Invalid argument

asin - Arc Sine of Angle

Interface

F_TYPE asin (F_TYPE x)

F_TYPE asind (F_TYPE x)

Description

asin( ) computes the principal value of the arc sine of x in the interval [-pi/2,pi/2] radians for x in the interval [-1,1].

asind( ) computes the principal value of the arc sine of x in the interval [-90,90] degrees for x in the interval [-1,1].

Exceptions

Exceptional Argument	Routine Behavior
|x| >1	Invalid argument

asinh - Hyperbolic Arc Sine of Angle

Interface

F_TYPE asinh (F_TYPE x)

Description

asinh( ) returns the hyperbolic arc sine of x for x in the interval [-infinity, +infinity]. asinh(x) = ln(x + sqrt(x**2 + 1)).

asinh( ) is the inverse function of sinh( ). asinh(sinh (x)) = x.

Exceptions

None.

atan - Arc Tangent of Angle with One Argument

Interface

F_TYPE atan (F_TYPE x)

F_TYPE atand (F_TYPE x)

Description

atan( ) computes the principal value of the arc tangent of x in the interval [-pi/2,pi/2] radians for x in the interval [-infinity, +infinity].

atand( ) computes the principal value of the arc tangent of x in the interval [-90,90] degrees for x in the interval [-infinity, +infinity].

Exceptions

None.

atan2 - Arc Tangent of Angle with Two Arguments

Interface

F_TYPE atan2 (F_TYPE y, F_TYPE x)

F_TYPE atand2 (F_TYPE y, F_TYPE x)

Description

atan2() computes the angle in the interval [-pi,pi] whose arc tangent is y/x radians for x and y in the interval [-infinity, +infinity]. The sign of atan2() is the same as the sign of y. The atan2(y, x) function is computed as follows, where f is the number of fraction bits associated with the data type:

Value of Input Arguments	Angle Returned
$x = 0$ or $y/x > 2^{f+1}$	$\pi/2 * (sign\,y)$
$x > 0$ and $y/x < 2^{f+1}$	$atan\,(y/x)$
$x < 0$ and $y/x \leq 2^{f+1}$	$\pi * (sign\,y) + atan\,(y/x)$

atand2() computes the angle in the interval [-180,180] whose arc tangent is y/x degrees for x and y in the interval [-infinity, +infinity]. The sign of atand2() is the same as the sign of y.

Exceptions

Exceptional Argument	Routine Behavior
y = x = 0	Invalid argument
|y| = infinity and |x| = infinity	Invalid argument

atanh - Hyperbolic Arc Tangent of Angle

Interface

F_TYPE atanh (F_TYPE x)

Description

atanh( ) returns the hyperbolic arc tangent of x for x in the interval (-1,1). atanh( ) is the inverse function of tanh( ). atanh(tanh (x)) = x.

atanh(x) is computed as 1/2 ln((1+x)/(1-x)).

Exceptions

Exceptional Argument	Routine Behavior
|x| > or = 1	Invalid argument

bessel - Bessel Functions

Interface

F_TYPE j0 (F_TYPE x)

F_TYPE j1 (F_TYPE x)

F_TYPE jn (int n, F_TYPE x)

F_TYPE y0 (F_TYPE x)

F_TYPE y1 (F_TYPE x)

F_TYPE yn (int n, F_TYPE x)

Description

j0( ) and j1( ) return the value of the Bessel function of the first kind of orders 0 and 1, respectively.

jn( ) returns the value of the Bessel function of the first kind of order n.

y0( ) and y1( ) return the value of the Bessel function of the second kind of orders 0 and 1, respectively.

yn( ) returns the value of the Bessel function of the second kind of order n.

The value of x must be positive for the y family of Bessel functions. The value of n specifies some integer value.

Exceptions

Exceptional Argument	Routine Behavior
(y0( ), y1( ), yn( )) x < 0	Invalid argument
(y0( ), y1( ), yn( )) x = 0	Overflow

The j1( ) and jn( ) functions can result in an underflow as x becomes small. The largest value of x for which this occurs is a function of n.

The y1( ) and yn( ) functions can result in an overflow as x becomes small. The largest value of x for which this occurs is a function of n.

cabs - Complex Absolute Value

Interface

F_TYPE cabs (F_TYPE x, F_TYPE y)

Description

cabs(x,y) is defined as the square root of (x**2 + y**2) and returns the same value as hypot(x,y).

Exceptions

Exceptional Argument	Routine Behavior
sqrt(x**2 + y**2) > max_float	Overflow

See Also

Appendix A, Critical Floating-Point Values

cbrt - Cube Root

Interface

F_TYPE cbrt (F_TYPE x)

Description

cbrt( ) returns the cube root of x.

Exceptions

None.

ccos - Cosine of Angle of a Complex Number

Interface

F_COMPLEX ccos (F_TYPE x, F_TYPE y)

Description

ccos( ) returns the cosine of a complex number, x + iy.

ccos(x,y) is defined as cos (x + iy) = (cos x * cosh y - i * sin x * sinh y).

Exceptions

Exceptional Argument	Routine Behavior
| x | = infinity	Invalid argument
(sin x sinh y) > max_float	Overflow
(cos x cosh y) > max_float	Overflow

See Also

Appendix A, Critical Floating-Point Values

cdiv - Complex Division

Interface

F_COMPLEX cdiv (F_TYPE a, F_TYPE b, F_TYPE c, F_TYPE d)

Description

cdiv( ) returns the quotient of two complex numbers: $(a + ib)/(c + id)$.

Exceptions

Exceptional Argument	Routine Behavior
c=d=0	Invalid argument

The quotient may overflow.

ceil - Ceiling

Interface

F_TYPE ceil (F_TYPE x)

Description

ceil( ) returns the smallest floating-point number of integral value greater than or equal to x.

Exceptions

None.

cexp - Complex Exponential

Interface

F_COMPLEX cexp (F_TYPE x, F_TYPE y)

Description

cexp( ) returns the exponential of a complex number.

cexp(x,y) is defined as e**(x + iy) = e**x cos y + ie**x sin y.

Exceptions

Exceptional Argument	Routine Behavior
| y| = infinity	Invalid argument
| e**x cos y| > max_float	Overflow
| e**x sin y| > max_float	Overflow

See Also

Appendix A, Critical Floating-Point Values

clog - Complex Natural Logarithm

Interface

F_COMPLEX clog (F_TYPE x, F_TYPE y)

Description

clog( ) returns the natural logarithm of a complex number.

clog(x,y) is defined as ln(x + iy) = 1/2 ln(x**2 + y**2) + i * atan2(y,x).

Exceptions

Exceptional Argument	Routine Behavior
y=x=0	Invalid argument
|y|=|x|=infinity	Invalid argument

cmul - Complex Multiplication

Interface

F_COMPLEX cmul (F_TYPE a, F_TYPE b, F_TYPE c, F_TYPE d)

Description

cmul( ) returns the product of two complex numbers.

cmul(a,b,c,d) is defined as (a + ib) * (c + id).

Exceptions

None.

copysign - Copy Sign

Interface

F_TYPE copysign (F_TYPE x, F_TYPE y)

Description

copysign() returns x with the same sign as y. IEEE Std 754 requires copysign(x,NaN) = +x or -x.

Exceptions

None.

cos - Cosine of Angle

Interface

F_TYPE cos (F_TYPE x)

F_TYPE cosd (F_TYPE x)

Description

cos( ) computes the cosine of x, measured in radians.

cosd( ) computes the cosine of x, measured in degrees.

Exceptions

Exceptional Argument	Routine Behavior
$\|x\| =$ infinity	Invalid argument

cosh - Hyperbolic Cosine of Angle

Interface

F_TYPE cosh (F_TYPE x)

Description

cosh( ) computes the hyperbolic cosine of x.

cosh(x) is defined as (exp(x) + exp(-x))/2.

Exceptions

Exceptional Argument	Routine Behavior
| x | > ln(2 * max_float)	Overflow

See Also

Appendix A, Critical Floating-Point Values

cot - Cotangent of Angle

Interface

F_TYPE cot (F_TYPE x)

F_TYPE cotd (F_TYPE x)

Description

cot( ) computes the cotangent of x, measured in radians.

cotd( ) computes the cotangent of x, measured in degrees.

Exceptions

Exceptional Argument	Routine Behavior
(cot) x=0	Overflow
(cotd) | x | = multiples of 180 degrees	Overflow

cpow - Complex Power

Interface

F_COMPLEX cpow (F_TYPE a, F_TYPE b, F_TYPE c, F_TYPE d)

Description

cpow( ) raises a complex base (a + ib) to a complex exponent (c + id).

cpow(a,b,c,d) is defined as e**((c + id) ln(a + ib)).

Exceptions

Exceptional Argument	Routine Behavior
sqrt (a**2 + b**2) > max_float	Overflow
c/2 * ln(a**2 + b**2) > max_float	Overflow
c/2 * ln(a**2 + b**2) - (d * atan2(b,c)) > max_float	Overflow
a=b=c=d=0	Invalid argument

See Also

Appendix A, Critical Floating-Point Values

csin - Sine of Angle of a Complex Number

Interface

F_COMPLEX csin (F_TYPE x, F_TYPE y)

Description

csin( ) computes the sine of a complex number, x + iy.

csin(x,y) is defined as csin (x + iy) = sin x * cosh y + i * cos x * sinh y.

Exceptions

Exceptional Argument	Routine Behavior
| x| = infinity	Invalid argument
| sin x * cosh y| > max_float	Overflow
| cos x * sinh y| > max_float	Overflow

See Also

Appendix A, Critical Floating-Point Values

csqrt - Complex Square Root

Interface

F_COMPLEX csqrt (F_TYPE x, F_TYPE y)

Description

csqrt( ) computes the square root of a complex number, x + iy. The root is chosen so that the real part of csqrt(x,y) is greater than or equal to zero.

Exceptions

None.

cvt_ftof - Convert Between Supported Floating-Point Data Types

Interface

int cvt_ftof void *x, int x_type, void *y, int y_type, options

Description

Note

This routine does not apply to OpenVMS Alpha. OpenVMS Alpha users should use the CVT$FTOF routine documented in the OpenVMS RTL Library (LIB$) Manual.

cvt_ftof( ) converts a floating-point value from one data type to another. x points to the input value to be converted, and y points to the converted result. The conversion is subject to the options specified in the options (bit field) argument.

x_type and y_type identify the data type of x and y as follows:

Values for x_type and y_type	Floating-Point Data Type
CVT_VAX_F	VAX F Floating (4 bytes)
CVT_VAX_D	VAX D Floating (8 bytes)
CVT_VAX_G	VAX G Floating (8 bytes)
CVT_VAX_H	VAX H Floating (16 bytes)
CVT_IEEE_S	IEEE Little Endian S Floating (4 bytes)
CVT_IEEE_T	IEEE Little Endian T Floating (8 bytes)
CVT_IEEE_X	IEEE Little Endian X Floating (16 bytes)
CVT_BIG_ENDIAN_IEEE_S	IEEE Big Endian S Floating (4 bytes)
CVT_BIG_ENDIAN_IEEE_T	IEEE Big Endian T Floating (8 bytes)
CVT_BIG_ENDIAN_IEEE_X	IEEE Big Endian X Floating (16 bytes)
CVT_IBM_SHORT	IBM_Short_Floating (4 bytes)
CVT_IBM_LONG	IBM_Long_Floating (8 bytes)
CVT_CRAY_SINGLE	CRAY_Floating (8 bytes)

Provide a zero (0) value to the options argument to select the default behavior or choose one or more options (status condition option, rounding options, "FORCE" options, CRAY and IBM options) from the tables below as the options argument. Specify only the options that apply to your conversion. A conflicting or incompatible options argument will be reported as an error (CVT_INVALID_OPTION).

Applicable Conversion	Status Condition Option	Description
All	CVT_REPORT_ALL	Report all applicable status conditions as the default. The reporting of recoverable status conditions is disabled by default when this option is not used.

Applicable Conversion	Rounding Options	Description
All	CVT_ROUND_TO_NEAREST	The default rounding option for conversions to IEEE data types. This IEEE Std. 754 rounding mode results in the representable output value nearest to the infinitely precise result. If the two nearest representable values are equally near, the one with its least significant bit zero is the result.
All	CVT_BIASED_ROUNDING	The default rounding option for conversions to non-IEEE data types. Performs "traditional" style rounding. This mode results in the representable output value nearest to the infinitely precise result. If the two nearest representable values are equally near, the result is the value with the largest magnitude.
All	CVT_ROUND_TO_ZERO	Round the output value toward zero (truncate).
All	CVT_ROUND_TO_POS	Round the output value toward positive infinity.
All	CVT_ROUND_TO_NEG	Round the output value toward negative infinity.

Applicable Conversion	"FORCE" Options	Description
All	CVT_FORCE_ALL_SPECIAL_VALUES	Apply all applicable "FORCE" options for the current conversion.
IEEE	CVT_FORCE_DENORM_TO_ZERO[1]	Force a denormalized IEEE output value to zero.
IEEE	CVT_FORCE_INF_TO_MAX_FLOAT[1]	Force a positive IEEE infinite output value to +max_float and force a negative IEEE infinite output value to -max_float.
IEEE or VAX	CVT_FORCE_INVALID_TO_ZERO[2]	Force an invalid IEEE NaN (not a number) output value or a VAX ROP (reserved operand) output value to zero.

[1]This option is valid only for conversions to IEEE output values.

[2]This option is valid only for conversions to IEEE or VAX output values.

Applicable Conversion	Options for CRAY Format Conversion	Description
CRAY	CVT_ALLOW_OVRFLW_RANGE_VALUES	Allow an input/output exponent value >60000 (8).
CRAY	CVT_ALLOW_UDRFLW_RANGE_VALUES	Allow an input/output exponent value <20000 (8).

Applicable Conversion	Option for IBM Format Conversion	Description
IBM	CVT_ALLOW_UNNORMALIZED_VALUES	Allow unnormalized input arguments. Allow an unnormalized output value for a small value that would normalize to zero.

Returns

The return value is a bit field containing the condition codes raised by the function. cvt_ftof( ) returns CVT_NORMAL; otherwise, it sets one or more of the following recoverable and unrecoverable conditions. Use the following condition names to determine which conditions are set:

Condition Name	Condition (Always reported by default)
CVT_INVALID_INPUT_TYPE	Invalid input type code.
CVT_INVALID_OUTPUT_TYPE	Invalid output type code.
CVT_INVALID_OPTION	Invalid option argument.

Condition Name	Condition (Only reported if the CVT_REPORT_ALL option is selected)
CVT_RESULT_INFINITE	Conversion produced an infinite result.[1]
CVT_RESULT_DENORMALIZED	Conversion produced a denormalized result.[1]
CVT_RESULT_OVERFLOW_RANGE	Conversion yielded an exponent >60000 (8).[2]
CVT_RESULT_UNDERFLOW_RANGE	Conversion yielded an exponent <20000 (8).[2]
CVT_RESULT_UNNORMALIZED	Conversion produced an unnormalized result.[3]
CVT_RESULT_INVALID	Conversion result is either ROP (reserved operand), NaN (not a number), or closest equivalent. CRAY and IBM data types return 0.[4]
CVT_RESULT_OVERFLOW	Conversion resulted in overflow.[4]
CVT_RESULT_UNDERFLOW	Conversion resulted in underflow.[4]

[1]For IEEE data type conversions.

[2]For CRAY data type conversions.

[3]For IBM data type conversions.

[4]For all data type conversions.

Condition Name	Condition (Only reported if the CVT_ REPORT_ALL option is selected)
CVT_RESULT_INEXACT	Conversion resulted in a loss of precision.[4]

[4]For all data type conversions.

See Also

Appendix A, Critical Floating-Point Values

ANSI/IEEE Std 754-1985, IEEE Standard for Binary Floating-Point Arithmetic

Example

```
status = cvt_ftof( &big_x, CVT_BIG_ENDIAN_IEEE_T,
                   &little_x, CVT_IEEE_T, 0 );
```

This example converts the value pointed to by big_x, which is of type IEEE Big Endian T Floating, to the IEEE Little Endian T Floating data type. It stores the result in the location pointed to by little_x. No conversion options are specified.

```
status = cvt_ftof(&x, CVT_VAX_D, &y, CVT_IEEE_T,
      (CVT_FORCE_ALL_SPECIAL_VALUES | CVT_REPORT_ALL) );
```

This example converts the value pointed to by x, which is of type VAX D Floating, to the IEEE Little Endian T Floating data type. It stores the result in the location pointed to by y. Any special IEEE values that would normally be generated will be removed. That is, NaN and Denormalized results will be returned as zero and infinite results will go to +- max_float. In addition, all recordable status conditions will be reported.

drem - Remainder

Interface

F_TYPE drem (F_TYPE x, F_TYPE y)

Description

drem( ) returns the remainder r = x-n*y, where n = rint(x/y). Additionally, if | n-x/y | =1/2, then n is even. The remainder is computed exactly, and | r | is less than or equal to | y | /2. The drem( ) and remainder( ) functions are aliases of each other.

Exceptions

Exceptional Argument	Routine Behavior
x = infinity	Invalid argument

Note that rem(x,0) has value 0 and is not an exceptional case.

erf - Error Functions

Interface

F_TYPE erf (F_TYPE x)

F_TYPE erfc (F_TYPE x)

Description

erf( ) returns the value of the error function. The definition of the erf( ) function is (2/sqrt(pi)) times the area under the curve exp(-t * t) between 0 and x.

erfc( ) returns (1.0-erf(x)).

Exceptions

The erfc( ) function can result in an underflow as x gets large.

exp - Exponential

Interface

F_TYPE exp (F_TYPE x)

F_TYPE expm1 (F_TYPE x)

Description

exp( ) computes the value of the exponential function, defined as e**x, where e is the constant used as a base for natural logarithms.

expm1( ) computes exp(x)-1 accurately, even for tiny x.

Exceptions

Exceptional Argument	Routine Behavior
x > ln(max_float)	Overflow
x < ln(min_float)	Underflow

See Also

Appendix A, Critical Floating-Point Values

fabs - Absolute Value

Interface

F_TYPE fabs (F_TYPE x)

Description

fabs( ) computes the absolute value of x.

Exceptions

None.

finite - Check for Finite Value

Interface

int finite (F_TYPE x)

Description

finite( ) returns the integer value 1 (true) or 0 (false).

finite(x) = 1 when -infinity < x < +infinity.

finite(x) = 0 when | x | = infinity or x is a NaN.

Exceptions

None.

floor - Floor

Interface

F_TYPE floor (F_TYPE x)

Description

floor() returns the largest floating-point number of integral value less than or equal to x.

Exceptions

None.

fmod - Modulo Remainder

Interface

F_TYPE fmod (F_TYPE x, F_TYPE y)

Description

fmod( ) computes the floating-point remainder of x modulo y. It returns the remainder r = x-n*y, where n = trunc(x/y). The remainder is computed exactly.

The result has the same sign as x and a magnitude less than the magnitude of y.

Exceptions

Exceptional Argument	Routine Behavior
x = infinity	Invalid argument

Note that fmod(x,0) has value 0 and is not an exceptional case.

fp_class - Classifies IEEE Floating-Point Values

Interface

int fp_class (F_TYPE x)

Description

These routines determine the class of IEEE floating-point values. They return one of the constants in the file <fp_class.h> and never cause an exception, even for signaling NaNs. These routines implement the recommended function class(x) in the appendix of the IEEE Std 754. The constants in <fp_class.h> refer to the following classes of values:

Constant	Class
FP_SNAN	Signaling NaN (Not-a-Number)
FP_QNAN	Quiet NaN (Not-a-Number)
FP_POS_INF	+Infinity
FP_NEG_INF	-Infinity
FP_POS_NORM	Positive normalized
FP_NEG_NORM	Negative normalized
FP_POS_DENORM	Positive denormalized
FP_NEG_DENORM	Negative denormalized
FP_POS_ZERO	+0.0 (positive zero)
FP_NEG_ZERO	-0.0 (negative zero)

Exceptions

None.

See Also

ANSI/IEEE Std 754-1985, IEEE Standard for Binary Floating-Point Arithmetic

frexp - Convert to Fraction and Integral Power of 2

Interface

F_TYPE frexp (F_TYPE x, int *n)

Description

frexp( ) breaks a floating-point number into a normalized fraction and an integral power of 2. It stores the integer in the int object pointed to by the n parameter and returns the fraction part.

Exceptions

None.

hypot - Euclidean Distance

Interface

F_TYPE hypot (F_TYPE x, F_TYPE y)

Description

hypot( ) computes the length of the hypotenuse of a right triangle, where x and y represent the perpendicular sides of the triangle.

hypot(x,y) is defined as the square root of (x**2 + y**2) and returns the same value as cabs(x,y).

Exceptions

Exceptional Argument	Routine Behavior
sqrt(x**2 + y**2) > max_float	Overflow

See Also

Appendix A, Critical Floating-Point Values

ilogb - Computes an Unbiased Exponent

Interface

int ilogb (F_TYPE x)

Description

ilogb(x) returns the unbiased exponent of x as an integer, (as if x were normalized >= 1.0 and < 2.0) except:

ilogb(NaN) is INT_MIN
ilogb(inf) is INT_MAX
logb(0) is INT_MIN

There are no errors. The sign of x is ignored.

Exceptions

None.

isnan - Check for NaN Value

Interface

int isnan (F_TYPE x)

Description

isnan( ) returns 1 (true) if x is NaN (the IEEE floating-point reserved Not-a-Number value) and 0 (false) otherwise.

Exceptions

None.

ldexp - Multiply by an Integral Power of 2

Interface

F_TYPE ldexp (F_TYPE x, int n)

Description

ldexp( ) multiplies a floating-point number, x, by 2**n.

Exceptions

Exceptional Argument	Routine Behavior
| x*(2**n)| > max_float	Overflow
| x*(2**n)| < min_float	Underflow

See Also

Appendix A, Critical Floating-Point Values

lgamma - Computes the Logarithm of the gamma Function

Interface

F_TYPE lgamma (F_TYPE x)

Description

lgamma( ) returns the logarithm of the absolute value of gamma of x, or ln(| G(x)|), where G is the gamma function. The sign of gamma of x is returned in the external integer variable signgam as +1 or -1. The x parameter cannot be 0 or a negative integer.

gamma( ) returns the natural log of the gamma function and so is functionally equivalent to lgamma( ). Because of this, gamma( ) is marked TO BE WITHDRAWN in the X/Open Portability Guide, Revision 4 (XPG4).

Exceptions

Exceptional Argument	Routine Behavior
| x| = infinity	Invalid argument
x = 0, -1, -2, -3, ...	Invalid argument
| x| > lgamma_max_float	Overflow

See Also

Appendix A, Critical Floating-Point Values

log - Logarithm Functions

Interface

F_TYPE ln (F_TYPE x)

F_TYPE log2 (F_TYPE x)

F_TYPE log10 (F_TYPE x)

F_TYPE log1p (F_TYPE y)

Description

ln( ) computes the natural (base e) logarithm of x.

log2( ) computes the base 2 logarithm of x.

log10( ) computes the common (base 10) logarithm of x.

log1p( ) computes ln(1+y) accurately, even for tiny y.

Exceptions

Exceptional Argument	Routine Behavior
$x < 0$	Invalid argument
$x = 0$	Overflow
$1+y < 0$	Invalid argument
$1+y = 0$	Overflow

logb - Radix-independent Exponent

Interface

F_TYPE logb (F_TYPE x)

Description

logb( ) returns a signed integer converted to double-precision floating-point and so chosen that 1 <= | x| /2**n < 2 unless x = 0 or | x| = infinity.

IEEE Std 754 defines logb(+infinity) = +infinity and logb(0) = -infinity. The latter is required to signal division by zero.

Exceptions

Exceptional Argument	Routine Behavior
x = 0	Invalid argument

modf - Return the Fractional Part and Integer Part of a Floating-Point Number

Interface

F_TYPE modf (F_TYPE x, F_TYPE *n)

Description

modf( ) splits a floating-point number x into a fractional part f and an integer part i such that $|f| < 1.0$ and $(f + i) = x$. Both f and i have the same sign as x. modf( ) returns f and stores i into the location pointed to by n.

Exceptions

None.

nextafter - Next Machine Number After

Interface

F_TYPE nextafter (F_TYPE x, F_TYPE y)

Description

nextafter( ) returns the machine-representable number next to x in the direction y.

Exceptions

Exceptional Argument	Routine Behavior
x = max_float and y = +infinity	Overflow
x = -max_float and y = -infinity	Overflow
x = min_float and y is less than or equal to 0	Underflow
x = -min_float and y is greater than or equal to 0	Underflow

See Also

ANSI/IEEE Std 754-1985, IEEE Standard for Binary Floating-Point Arithmetic

nint - Round to the Nearest Integer

Interface

F_TYPE nint (F_TYPE x)

Description

nint( ) returns the nearest integral value to x, except halfway cases are rounded to the integral value larger in magnitude. This function corresponds to the Fortran generic intrinsic function nint( ).

Exceptions

None.

pow - Raise the Base to a Floating-Point Exponent

Interface

F_TYPE pow (F_TYPE x, F_TYPE y)

Description

pow( ) raises a floating-point base x to a floating-point exponent y. The value of pow(x,y) is computed as e**(y ln(x)) for positive x. If x is 0 or negative, see your language reference manual.

Passing a NaN input value to pow( ) produces a NaN result for nonzero values of y. For pow(NaN,0), see your language reference manual.

Exceptions

Exceptional Argument	Routine Behavior
y ln(x) > ln(max_float)	Overflow
y ln(x) < ln(min_float)	Underflow

Fortran-Exceptional Argument	Routine Behavior
x < 0	Invalid argument
x = 0 and y < 0	Invalid argument
x = 0 and y = 0	Invalid argument
x = +infinity and y = 0	Invalid argument
x = 1 and |y| = infinity	Invalid argument

ANSI C-Exceptional Argument	Routine Behavior
|x| = 1 and |y| = infinity	Invalid argument
x < 0 and y is not integral	Invalid argument

See Also

Appendix A, Critical Floating-Point Values

random - Random Number Generator, Uniformly Distributed

Interface

F_TYPE random (int *n)

Description

random( ) is a general random number generator. The argument to the random function is an integer passed by reference. There are no restrictions on the input argument, although it should be initialized to different values on separate runs in order to obtain different random sequences. This function must be called again to obtain the next pseudo random number. The argument is updated automatically.

The result is a floating-point number that is uniformly distributed in the interval (0.0,1.0).

Exceptions

None.

remainder - Remainder

Interface

F_TYPE remainder (F_TYPE x, F_TYPE y)

Description

remainder( ) returns the remainder r = x-n*y, where n = rint(x/y). Additionally, if | n-x/y| = 1/2, then n is even. Consequently, the remainder is computed exactly, and | r| is less than or equal to | y| /2. The drem( ) and remainder( ) functions are aliases of each other.

Exceptions

Exceptional Argument	Routine Behavior
x = infinity	Invalid argument

Note that rem(x,0) has value 0 and is not an exceptional case.

rint - Return the Nearest Integral Value

Interface

F_TYPE rint (F_TYPE x)

Description

rint() rounds x to an integral value according to the current IEEE rounding direction specified by the user.

Exceptions

None.

scalb - Exponent Adjustment

Interface

F_TYPE scalb (F_TYPE x, F_TYPE y)

Description

scalb( ) = x*(2**y) computed, for integer-valued floating point number y.

Exceptions

Exceptional Argument	Routine Behavior
x*(2**y) > max_float	Overflow
x*(2**y) < min_float	Underflow
x=0, y=infinity	Invalid argument
x=infinity, y=-infinity	Invalid argument

See Also

Appendix A, Critical Floating-Point Values

sin - Sine of Angle

Interface

F_TYPE sin (F_TYPE x)

F_TYPE sind (F_TYPE x)

Description

sin() computes the sine of x, measured in radians.

sind() computes the sine of x, measured in degrees.

Exceptions

Exceptional Argument	Routine Behavior
$\|x\| = \text{infinity}$	Invalid argument
(sind) $\|x\| < (180/\text{pi}) * \text{min\_float}$	Underflow

See Also

Appendix A, Critical Floating-Point Values

sincos - Sine and Cosine of Angle

Interface

F_COMPLEX sincos (F_TYPE x)

F_COMPLEX sincosd (F_TYPE x)

Description

sincos( ) computes both the sine and cosine of x, measured in radians.

sincosd( ) computes both the sine and cosine of x, measured in degrees.

sincos(x) is defined as (sin x + icos y).

Exceptions

Exceptional Argument	Routine Behavior
|x| = infinity	Invalid argument
(sind) |x| < (180/pi) * min_float	Underflow

sinh - Hyperbolic Sine

Interface

F_TYPE sinh (F_TYPE x)

Description

sinh( ) computes the hyperbolic sine of x.

sinh(x) is defined as (exp(x)-exp(-x))/2.

Exceptions

Exceptional Argument	Routine Behavior
$\|x\| > \ln(2 * max\_float)$	Overflow

See Also

Appendix A, Critical Floating-Point Values

sinhcosh - Hyperbolic Sine and Cosine

Interface

F_COMPLEX sinhcosh (F_TYPE x)

Description

sinhcosh( ) computes both the hyperbolic sine and hyperbolic cosine of x. sinhcosh(x) is defined as (sinh x + icosh x).

Exceptions

Exceptional Argument	Routine Behavior
| x | > ln(2 * max_float)	Overflow

See Also

Appendix A, Critical Floating-Point Values

sqrt - Square Root

Interface

F_TYPE sqrt (F_TYPE x)

Description

sqrt( ) computes the rounded square root of x.

For platforms supporting a signed zero, sqrt(-0) = 0.

Exceptions

Exceptional Argument	Routine Behavior
x < 0	Invalid argument

tan - Tangent of Angle

Interface

F_TYPE tan (F_TYPE x)

F_TYPE tand (F_TYPE x)

Description

tan( ) computes the tangent of x, measured in radians.

tand( ) computes the tangent of x, measured in degrees.

Exceptions

Exceptional Argument	Routine Behavior
| x| = infinity	Invalid argument
(tand) | x| < (180/pi) * min_float	Underflow
(tand) x = (2n+1) * 90	Overflow

See Also

Appendix A, Critical Floating-Point Values

tanh - Hyperbolic Tangent

Interface

F_TYPE tanh (F_TYPE x)

Description

tanh() computes the hyperbolic tangent of x.

tanh(x) is defined as (exp(x)-exp(-x))/(exp(x) + exp(-x)).

Exceptions

None.

trunc - Truncation

Interface

F_TYPE trunc (F_TYPE x)

Description

trunc( ) truncates x to an integral value.

Exceptions

None.

unordered - Check for x Unordered with Respect to y

Interface

int unordered (F_TYPE x, F_TYPE y)

Description

unordered(x,y) returns the value 1 (true) if x, y, or both are a NaN and returns the value 0 (false) otherwise.

Exceptions

None.

A

Critical Floating-Point Values

Table A–1 contains the hexadecimal and decimal boundary values used in CPML calculations and exception checking.

Table A–1 Hexadecimal and Decimal Boundary Values

Data Type	Value for: max_float
F	Hexadecimal: FFFF7FFF
G	Hexadecimal: FFFFFFFFFFFF7FFF
S	Hexadecimal: 7F7FFFFF
T	Hexadecimal: 7FEFFFFFFFFFFFFF
X	Hexadecimal: 7FFEFFFFFFFFFFFFFFFFFFFFFFFFFFFF
F	Decimal: 1.701411e38
G	Decimal: 8.988465674311579e307
S	Decimal: 3.402823e38
T	Decimal: 1.797693134862316e308
X	Decimal: 1.189731495357231765085759326628007016196477e4932

Data Type	Value for: min_float
F	Hexadecimal: 00000080
G	Hexadecimal: 0000000000000010
S	Hexadecimal: 00000001
T	Hexadecimal: 0000000000000001
X	Hexadecimal: 00000000000000000000000000000001
F	Decimal: 2.9387359e-39
G	Decimal: 5.562684646268003e-309
S	Decimal: 1.4012985e-45
T	Decimal: 4.940656458412465e-324
X	Decimal: 6.475175119438025110924438958227646552499 6e-4966

(continued on next page)

Table A–1 (Cont.) Hexadecimal and Decimal Boundary Values

Data Type	Value for: ln(max_float)
F	Hexadecimal: 0F3443B0
G	Hexadecimal: 7B616E3A28B740A6
S	Hexadecimal: 42B17218
T	Hexadecimal: 40862E42FEFA39EF
X	Hexadecimal: 400C62E42FEFA39EF35793C7673007E6
F	Decimal: 88.029692
G	Decimal: 709.0895657128241
S	Decimal: 88.7228391
T	Decimal: 709.7827128933840
X	Decimal: 11356.523406294143949491931077970764891 2527

Data Type	Value for: ln(min_float)
F	Hexadecimal: 7218C3B1
G	Hexadecimal: 39EFFEFA2E42C0A6
S	Hexadecimal: C2CE8ED0
T	Hexadecimal: C0874385446D71C3
X	Hexadecimal: C00C6546282207802C89D24D65E96274
F	Decimal: -88.72284
G	Decimal: -709.7827128933840
S	Decimal: -103.2789
T	Decimal: -744.4400719213813
X	Decimal: -11432.769596155737933527826611331164313 8373

(continued on next page)

Table A–1 (Cont.) Hexadecimal and Decimal Boundary Values

Data Type	Value for: ln(2 * max_float)
F	Hexadecimal: 721843B1
G	Hexadecimal: 39EFFEFA2E4240A6
S	Hexadecimal: 42B2D4FC
T	Hexadecimal: 408633CE8FB9F87E
X	Hexadecimal: 400C62E9BB80635D81D36125B64DA4A6
F	Decimal: 88.72284
G	Decimal: 709.7827128933840
S	Decimal: 89.41599
T	Decimal: 710.4758600739439
X	Decimal: 11357.2165534747038948013483100922230678208

Data Type	Value for: (180/pi) * min_float
F	Hexadecimal: 2EE10365
G	Hexadecimal: C1F81A63A5DC006C
S	Hexadecimal: 00000039
T	Hexadecimal: 0000000000000039
X	Hexadecimal: 00000000000000000000000000000039
F	Decimal: 1.683772e-37
G	Decimal: 3.187183529933798e-307
S	Decimal: 8.028849e-44
T	Decimal: 2.830787630910868e-322
X	Decimal: 3.71000205951917569316937757202433432154392e-4964

(continued on next page)

Table A–1 (Cont.) Hexadecimal and Decimal Boundary Values

Data Type	Value for: lgamma_max_float
F	Hexadecimal: 50F97CC6
G	Hexadecimal: F55FC5015ABD7F67
S	Hexadecimal: 7BC650F9
T	Hexadecimal: 7F475ABDC501F55F
X	Hexadecimal: 7FF171AA9917FFFBD7EA44AE6D203DF6
F	Decimal: 2.0594342e36
G	Decimal: 1.2812545499066958e305
S	Decimal: 2.0594342e36
T	Decimal: 1.2812545499066958e305
X	Decimal: 1.0485738685148938358098967157129705040168e4928

B

CPML Entry-Point Names

Each entry-point name in Table B–1 is unique and corresponds to data-type specific calculations in a CPML routine. For example, the acos function has five entry-point-names for the OpenVMS Alpha operating system. Because five floating-point data types are available, five acos routines are provided: math$acos_s, math$acos_t, math$acos_f, math$acos_g, and math$acos_x. Use the entry-point name that corresponds to your input argument data type.

Table B–1 Entry-Point Names for CPML Platforms

		Entry-Point Names	
Generic Function Name	Data Type Required	OpenVMS Alpha	Compaq Tru64 UNIX Alpha
acos	S_FLOAT	math$acos_s	acosf
	T_FLOAT	math$acos_t	acos
	X_FLOAT	math$acos_x	acosl
	F_FLOAT	math$acos_f	
	G_FLOAT	math$acos_g	
acosd	S_FLOAT	math$acosd_s	acosdf
	T_FLOAT	math$acosd_t	acosd
	X_FLOAT	math$acosd_x	acosdl
	F_FLOAT	math$acosd_f	
	G_FLOAT	math$acosd_g	
acosh	S_FLOAT	math$acosh_s	acoshf
	T_FLOAT	math$acosh_t	acosh
	X_FLOAT	math$acosh_x	acoshl
	F_FLOAT	math$acosh_f	
	G_FLOAT	math$acosh_g	

(continued on next page)

Table B–1 (Cont.) Entry-Point Names for CPML Platforms

Generic Function Name	Data Type Required	Entry-Point Names: OpenVMS Alpha	Entry-Point Names: Compaq Tru64 UNIX Alpha
asin	S_FLOAT	math$asin_s	asinf
	T_FLOAT	math$asin_t	asin
	X_FLOAT	math$asin_x	asinl
	F_FLOAT	math$asin_f	
	G_FLOAT	math$asin_g	
asind	S_FLOAT	math$asind_s	asindf
	T_FLOAT	math$asind_t	asind
	X_FLOAT	math$asind_x	asindl
	F_FLOAT	math$asind_f	
	G_FLOAT	math$asind_g	
asinh	S_FLOAT	math$asinh_s	asinhf
	T_FLOAT	math$asinh_t	asinh
	X_FLOAT	math$asinh_x	asinhl
	F_FLOAT	math$asinh_f	
	G_FLOAT	math$asinh_g	
atan	S_FLOAT	math$atan_s	atanf
	T_FLOAT	math$atan_t	atan
	X_FLOAT	math$atan_x	atanl
	F_FLOAT	math$atan_f	
	G_FLOAT	math$atan_g	
atan2	S_FLOAT	math$atan2_s	atan2f
	T_FLOAT	math$atan2_t	atan2
	X_FLOAT	math$atan2_x	atan2l
	F_FLOAT	math$atan2_f	
	G_FLOAT	math$atan2_g	
atand	S_FLOAT	math$atand_s	atandf
	T_FLOAT	math$atand_t	atand
	X_FLOAT	math$atand_x	atandl
	F_FLOAT	math$atand_f	
	G_FLOAT	math$atand_g	

(continued on next page)

Table B–1 (Cont.) Entry-Point Names for CPML Platforms

Generic Function Name	Data Type Required	Entry-Point Names: OpenVMS Alpha	Entry-Point Names: Compaq Tru64 UNIX Alpha
atand2	S_FLOAT	math$atand2_s	atand2f
	T_FLOAT	math$atand2_t	atand2
	X_FLOAT	math$atand2_x	atand2l
	F_FLOAT	math$atand2_f	
	G_FLOAT	math$atand2_g	
atanh	S_FLOAT	math$atanh_s	atanhf
	T_FLOAT	math$atanh_t	atanh
	X_FLOAT	math$atanh_x	atanhl
	F_FLOAT	math$atanh_f	
	G_FLOAT	math$atanh_g	
cabs	S_FLOAT	math$hypot_s	cabsf
	T_FLOAT	math$hypot_t	cabs
	X_FLOAT	math$hypot_x	cabsl
	F_FLOAT	math$hypot_f	
	G_FLOAT	math$hypot_g	
cbrt	S_FLOAT	math$cbrt_s	cbrtf
	T_FLOAT	math$cbrt_t	cbrt
	X_FLOAT	math$cbrt_x	cbrtl
	F_FLOAT	math$cbrt_f	
	G_FLOAT	math$cbrt_g	
ccos	S_FLOAT	math$ccos_s	ccosf
	T_FLOAT	math$ccos_t	ccos
	X_FLOAT	math$ccos_x	ccosl
	F_FLOAT	math$ccos_f	
	G_FLOAT	math$ccos_g	

(continued on next page)

Table B–1 (Cont.) Entry-Point Names for CPML Platforms

Generic Function Name	Data Type Required	Entry-Point Names: OpenVMS Alpha	Entry-Point Names: Compaq Tru64 UNIX Alpha
cdiv	S_FLOAT	math$cdiv_s	cdivf
	T_FLOAT	math$cdiv_t	cdiv
	X_FLOAT	math$cdiv_x	cdivl
	F_FLOAT	math$cdiv_f	
	G_FLOAT	math$cdiv_g	
ceil	S_FLOAT	math$ceil_s	ceilf
	T_FLOAT	math$ceil_t	ceil
	X_FLOAT	math$ceil_x	ceill
	F_FLOAT	math$ceil_f	
	G_FLOAT	math$ceil_g	
cexp	S_FLOAT	math$cexp_s	cexpf
	T_FLOAT	math$cexp_t	cexp
	X_FLOAT	math$cexp_x	cexpl
	F_FLOAT	math$cexp_f	
	G_FLOAT	math$cexp_g	
clog	S_FLOAT	math$clog_s	clogf
	T_FLOAT	math$clog_t	clog
	X_FLOAT	math$clog_x	clogl
	F_FLOAT	math$clog_f	
	G_FLOAT	math$clog_g	
cmul	S_FLOAT	math$cmul_s	cmulf
	T_FLOAT	math$cmul_t	cmul
	X_FLOAT	math$cmul_x	cmull
	F_FLOAT	math$cmul_f	
	G_FLOAT	math$cmul_g	
copysign	S_FLOAT	math$copysign_s	copysignf
	T_FLOAT	math$copysign_t	copysign
	X_FLOAT	math$copysign_x	copysignl
	F_FLOAT	math$copysign_f	
	G_FLOAT	math$copysign_g	

(continued on next page)

Table B–1 (Cont.) Entry-Point Names for CPML Platforms

Generic Function Name	Data Type Required	Entry-Point Names: OpenVMS Alpha	Entry-Point Names: Compaq Tru64 UNIX Alpha
cos	S_FLOAT	math$cos_s	cosf
	T_FLOAT	math$cos_t	cos
	X_FLOAT	math$cos_x	cosl
	F_FLOAT	math$cos_f	
	G_FLOAT	math$cos_g	
cosd	S_FLOAT	math$cosd_s	cosdf
	T_FLOAT	math$cosd_t	cosd
	X_FLOAT	math$cosd_x	cosdl
	F_FLOAT	math$cosd_f	
	G_FLOAT	math$cosd_g	
cosh	S_FLOAT	math$cosh_s	coshf
	T_FLOAT	math$cosh_t	cosh
	X_FLOAT	math$cosh_x	coshl
	F_FLOAT	math$cosh_f	
	G_FLOAT	math$cosh_g	
cot	S_FLOAT	math$cot_s	cotf
	T_FLOAT	math$cot_t	cot
	X_FLOAT	math$cot_x	cotl
	F_FLOAT	math$cot_f	
	G_FLOAT	math$cot_g	
cotd	S_FLOAT	math$cotd_s	cotdf
	T_FLOAT	math$cotd_t	cotd
	X_FLOAT	math$cotd_x	cotdl
	F_FLOAT	math$cotd_f	
	G_FLOAT	math$cotd_g	

(continued on next page)

Table B–1 (Cont.) Entry-Point Names for CPML Platforms

Generic Function Name	Data Type Required	Entry-Point Names: OpenVMS Alpha	Entry-Point Names: Compaq Tru64 UNIX Alpha
cpow	S_FLOAT	math$cpow_s	cpowf
	T_FLOAT	math$cpow_t	cpow
	X_FLOAT	math$cpow_x	cpowl
	F_FLOAT	math$cpow_f	
	G_FLOAT	math$cpow_g	
csin	S_FLOAT	math$csin_s	csinf
	T_FLOAT	math$csin_t	csin
	X_FLOAT	math$csin_x	csinl
	F_FLOAT	math$csin_f	
	G_FLOAT	math$csin_g	
csqrt	S_FLOAT	math$csqrt_s	csqrtf
	T_FLOAT	math$csqrt_t	csqrt
	X_FLOAT	math$csqrt_x	csqrtl
	F_FLOAT	math$csqrt_f	
	G_FLOAT	math$csqrt_g	
cvt_ftof	All supported types		cvt_ftof
drem	S_FLOAT	math$rem_s	dremf
	T_FLOAT	math$rem_t	drem
	X_FLOAT	math$rem_x	dreml
	F_FLOAT	math$rem_f	
	G_FLOAT	math$rem_g	
erf	S_FLOAT	math$erf_s	erff
	T_FLOAT	math$erf_t	erf
	X_FLOAT	math$erf_x	erfl
	F_FLOAT	math$erf_f	
	G_FLOAT	math$erf_g	

(continued on next page)

Table B–1 (Cont.) Entry-Point Names for CPML Platforms

Generic Function Name	Data Type Required	Entry-Point Names: OpenVMS Alpha	Entry-Point Names: Compaq Tru64 UNIX Alpha
erfc	S_FLOAT	math$erfc_s	erfcf
	T_FLOAT	math$erfc_t	erfc
	X_FLOAT	math$erfc_x	erfcl
	F_FLOAT	math$erfc_f	
	G_FLOAT	math$erfc_g	
exp	S_FLOAT	math$exp_s	expf
	T_FLOAT	math$exp_t	exp
	X_FLOAT	math$exp_x	expl
	F_FLOAT	math$exp_f	
	G_FLOAT	math$exp_g	
expm1	S_FLOAT	math$expm1_s	expm1f
	T_FLOAT	math$expm1_t	expm1
	X_FLOAT	math$expm1_x	expm1l
	F_FLOAT	math$expm1_f	
	G_FLOAT	math$expm1_g	
fabs	S_FLOAT	math$fabs_s	fabsf
	T_FLOAT	math$fabs_t	fabs
	X_FLOAT	math$fabs_x	fabsl
	F_FLOAT	math$fabs_f	
	G_FLOAT	math$fabs_g	
finite	S_FLOAT	math$finite_s	finitef
	T_FLOAT	math$finite_t	finite
	X_FLOAT	math$finite_x	finitel
	F_FLOAT	math$finite_f	
	G_FLOAT	math$finite_g	

(continued on next page)

Table B–1 (Cont.) Entry-Point Names for CPML Platforms

Generic Function Name	Data Type Required	Entry-Point Names: OpenVMS Alpha	Entry-Point Names: Compaq Tru64 UNIX Alpha
floor	S_FLOAT	math$floor_s	floorf
	T_FLOAT	math$floor_t	floor
	X_FLOAT	math$floor_x	floorl
	F_FLOAT	math$floor_f	
	G_FLOAT	math$floor_g	
fmod	S_FLOAT	math$mod_s	fmodf
	T_FLOAT	math$mod_t	fmod
	X_FLOAT	math$mod_x	fmodl
	F_FLOAT	math$mod_f	
	G_FLOAT	math$mod_g	
fp_class	S_FLOAT	math$fp_class_s	fp_classf
	T_FLOAT	math$fp_class_t	fp_class
	X_FLOAT	math$fp_class_x	fp_classl
	F_FLOAT	math$fp_class_f	
	G_FLOAT	math$fp_class_g	
frexp	S_FLOAT	math$frexp_s	frexpf
	T_FLOAT	math$frexp_t	frexp
	X_FLOAT	math$frexp_x	frexpl
	F_FLOAT	math$frexp_f	
	G_FLOAT	math$frexp_g	
hypot	S_FLOAT	math$hypot_s	hypotf
	T_FLOAT	math$hypot_t	hypot
	X_FLOAT	math$hypot_x	hypotl
	F_FLOAT	math$hypot_f	
	G_FLOAT	math$hypot_g	
ilogb	S_FLOAT	math$ilogb_s	ilogbf
	T_FLOAT	math$ilogb_t	ilogb
	X_FLOAT	math$ilogb_x	ilogbl
	F_FLOAT	math$ilogb_f	
	G_FLOAT	math$ilogb_g	

(continued on next page)

Table B–1 (Cont.) Entry-Point Names for CPML Platforms

Generic Function Name	Data Type Required	Entry-Point Names: OpenVMS Alpha	Entry-Point Names: Compaq Tru64 UNIX Alpha
isnan	S_FLOAT	math$isnan_s	isnanf
	T_FLOAT	math$isnan_t	isnan
	X_FLOAT	math$isnan_x	isnanl
	F_FLOAT	math$isnan_f	
	G_FLOAT	math$isnan_g	
j0	S_FLOAT	math$j0_s	j0f
	T_FLOAT	math$j0_t	j0
	X_FLOAT	math$j0_x	j0l
	F_FLOAT	math$j0_f	
	G_FLOAT	math$j0_g	
j1	S_FLOAT	math$j1_s	j1f
	T_FLOAT	math$j1_t	j1
	X_FLOAT	math$j1_x	j1l
	F_FLOAT	math$j1_f	
	G_FLOAT	math$j1_g	
jn	S_FLOAT	math$jn_s	jnf
	T_FLOAT	math$jn_t	jn
	X_FLOAT	math$jn_x	jnl
	F_FLOAT	math$jn_f	
	G_FLOAT	math$jn_g	
ldexp	S_FLOAT	math$ldexp_s	ldexpf
	T_FLOAT	math$ldexp_t	ldexp
	X_FLOAT	math$ldexp_x	ldexpl
	F_FLOAT	math$ldexp_f	
	G_FLOAT	math$ldexp_g	

(continued on next page)

Table B–1 (Cont.) Entry-Point Names for CPML Platforms

Generic Function Name	Data Type Required	Entry-Point Names: OpenVMS Alpha	Entry-Point Names: Compaq Tru64 UNIX Alpha
lgamma	S_FLOAT	math$lgamma_s	lgammaf
	T_FLOAT	math$lgamma_t	lgamma
	X_FLOAT	math$lgamma_x	lgammal
	F_FLOAT	math$lgamma_f	
	G_FLOAT	math$lgamma_g	
ln	S_FLOAT	math$ln_s	logf
	T_FLOAT	math$ln_t	log
	X_FLOAT	math$ln_x	logl
	F_FLOAT	math$ln_f	
	G_FLOAT	math$ln_g	
log2	S_FLOAT	math$log2_s	log2f
	T_FLOAT	math$log2_t	log2
	X_FLOAT	math$log2_x	log2l
	F_FLOAT	math$log2_f	
	G_FLOAT	math$log2_g	
log10	S_FLOAT	math$log10_s	log10f
	T_FLOAT	math$log10_t	log10
	X_FLOAT	math$log10_x	log10l
	F_FLOAT	math$log10_f	
	G_FLOAT	math$log10_g	
log1p	S_FLOAT	math$log1p_s	log1pf
	T_FLOAT	math$log1p_t	log1p
	X_FLOAT	math$log1p_x	log1pl
	F_FLOAT	math$log1p_f	
	G_FLOAT	math$log1p_g	
logb	S_FLOAT	math$logb_s	logbf
	T_FLOAT	math$logb_t	logb
	X_FLOAT	math$logb_x	logbl
	F_FLOAT	math$logb_f	
	G_FLOAT	math$logb_g	

(continued on next page)

Table B–1 (Cont.) Entry-Point Names for CPML Platforms

Generic Function Name	Data Type Required	Entry-Point Names: OpenVMS Alpha	Entry-Point Names: Compaq Tru64 UNIX Alpha
modf	S_FLOAT	math$modf_s	modff
	T_FLOAT	math$modf_t	modf
	X_FLOAT	math$modf_x	modfl
	F_FLOAT	math$modf_f	
	G_FLOAT	math$modf_g	
nextafter	S_FLOAT	math$nextafter_s	nextafterf
	T_FLOAT	math$nextafter_t	nextafter
	X_FLOAT	math$nextafter_x	nextafterl
	F_FLOAT	math$nextafter_f	
	G_FLOAT	math$nextafter_g	
nint	S_FLOAT	math$nint_s	nintf
	T_FLOAT	math$nint_t	nint
	X_FLOAT	math$nint_x	nintl
	F_FLOAT	math$nint_f	
	G_FLOAT	math$nint_g	
pow	S_FLOAT	math$pow_ss	powf
	T_FLOAT	math$pow_tt	pow
	X_FLOAT	math$pow_xx	powl
	F_FLOAT	math$pow_ff	
	G_FLOAT	math$pow_gg	
random	S_FLOAT	math$random_l_s	
	T_FLOAT		
	X_FLOAT		
	F_FLOAT	math$random_l_f	
	G_FLOAT		

(continued on next page)

Table B–1 (Cont.) Entry-Point Names for CPML Platforms

Generic Function Name	Data Type Required	Entry-Point Names: OpenVMS Alpha	Entry-Point Names: Compaq Tru64 UNIX Alpha
remainder	S_FLOAT	math$rem_s	remainderf
	T_FLOAT	math$rem_t	remainder
	X_FLOAT	math$rem_x	remainderl
	F_FLOAT	math$rem_f	
	G_FLOAT	math$rem_g	
rint	S_FLOAT	math$rint_s	rintf
	T_FLOAT	math$rint_t	rint
	X_FLOAT	math$rint_x	rintl
	F_FLOAT	math$rint_f	
	G_FLOAT	math$rint_g	
scalb	S_FLOAT	math$scalb_s	scalbf
	T_FLOAT	math$scalb_t	scalb
	X_FLOAT	math$scalb_x	scalbl
	F_FLOAT	math$scalb_f	
	G_FLOAT	math$scalb_g	
sin	S_FLOAT	math$sin_s	sinf
	T_FLOAT	math$sin_t	sin
	X_FLOAT	math$sin_x	sinl
	F_FLOAT	math$sin_f	
	G_FLOAT	math$sin_g	
sincos	S_FLOAT	math$sincos_s	sincosf
	T_FLOAT	math$sincos_t	sincos
	X_FLOAT	math$sincos_x	sincosl
	F_FLOAT	math$sincos_f	
	G_FLOAT	math$sincos_g	
sincosd	S_FLOAT	math$sincosd_s	sincosdf
	T_FLOAT	math$sincosd_t	sincosd
	X_FLOAT	math$sincosd_x	sincosdl
	F_FLOAT	math$sincosd_f	
	G_FLOAT	math$sincosd_g	

(continued on next page)

Table B–1 (Cont.) Entry-Point Names for CPML Platforms

Generic Function Name	Data Type Required	Entry-Point Names: OpenVMS Alpha	Entry-Point Names: Compaq Tru64 UNIX Alpha
sind	S_FLOAT	math$sind_s	sindf
	T_FLOAT	math$sind_t	sind
	X_FLOAT	math$sind_x	sindl
	F_FLOAT	math$sind_f	
	G_FLOAT	math$sind_g	
sinh	S_FLOAT	math$sinh_s	sinhf
	T_FLOAT	math$sinh_t	sinh
	X_FLOAT	math$sinh_x	sinhl
	F_FLOAT	math$sinh_f	
	G_FLOAT	math$sinh_g	
sinhcosh	S_FLOAT	math$sinhcosh_s	sinhcoshf
	T_FLOAT	math$sinhcosh_t	sinhcosh
	X_FLOAT	math$sinhcosh_x	sinhcoshl
	F_FLOAT	math$sinhcosh_f	
	G_FLOAT	math$sinhcosh_g	
sqrt	S_FLOAT	math$sqrt_s	sqrtf
	T_FLOAT	math$sqrt_t	sqrt
	X_FLOAT	math$sqrt_x	sqrtl
	F_FLOAT	math$sqrt_f	
	G_FLOAT	math$sqrt_g	
tan	S_FLOAT	math$tan_s	tanf
	T_FLOAT	math$tan_t	tan
	X_FLOAT	math$tan_x	tanl
	F_FLOAT	math$tan_f	
	G_FLOAT	math$tan_g	

(continued on next page)

Table B–1 (Cont.) Entry-Point Names for CPML Platforms

Generic Function Name	Data Type Required	Entry-Point Names: OpenVMS Alpha	Entry-Point Names: Compaq Tru64 UNIX Alpha
tand	S_FLOAT	math$tand_s	tandf
	T_FLOAT	math$tand_t	tand
	X_FLOAT	math$tand_x	tandl
	F_FLOAT	math$tand_f	
	G_FLOAT	math$tand_g	
tanh	S_FLOAT	math$tanh_s	tanhf
	T_FLOAT	math$tanh_t	tanh
	X_FLOAT	math$tanh_x	tanhl
	F_FLOAT	math$tanh_f	
	G_FLOAT	math$tanh_g	
trunc	S_FLOAT	math$trunc_s	truncf
	T_FLOAT	math$trunc_t	trunc
	X_FLOAT	math$trunc_x	truncl
	F_FLOAT	math$trunc_f	
	G_FLOAT	math$trunc_g	
unordered	S_FLOAT	math$unordered_s	unorderedf
	T_FLOAT	math$unordered_t	unordered
	X_FLOAT	math$unordered_x	unorderedl
	F_FLOAT	math$unordered_f	
	G_FLOAT	math$unordered_g	
y0	S_FLOAT	math$y0_s	y0f
	T_FLOAT	math$y0_t	y0
	X_FLOAT	math$y0_x	y0l
	F_FLOAT	math$y0_f	
	G_FLOAT	math$y0_g	
y1	S_FLOAT	math$y1_s	y1f
	T_FLOAT	math$y1_t	y1
	X_FLOAT	math$y1_x	y1l
	F_FLOAT	math$y1_f	
	G_FLOAT	math$y1_g	

(continued on next page)

Table B–1 (Cont.) Entry-Point Names for CPML Platforms

Generic Function Name	Data Type Required	Entry-Point Names: OpenVMS Alpha	Entry-Point Names: Compaq Tru64 UNIX Alpha
yn	S_FLOAT	math$yn_s	ynf
	T_FLOAT	math$yn_t	yn
	X_FLOAT	math$yn_x	ynl
	F_FLOAT	math$yn_f	
	G_FLOAT	math$yn_g	

Glossary

This glossary defines mathematical terms and symbolic names used in this manual.

complex number

See F_COMPLEX.

denormalized number

A floating-point number with a value very close to zero.

domain error

An exception condition resulting from passing an argument whose value is outside the range of permissible values.

exceptional argument

Any argument value passed to a CPML routine that does not return a meaningful result, or an argument defined differently for different platforms.

F_COMPLEX

A complex number identifier. F_COMPLEX indicates that a given routine returns two different values of the same floating-point data type. See Table 1–2 for more information.

F_TYPE

A floating-point number identifier. F_TYPE is used when it is not necessary to distinguish between the floating types. See Table 1–1 for more information.

floating-point number

See F_TYPE.

HUGE_RESULT

For VAX data types, HUGE_RESULT = max_float.

For IEEE data types, HUGE_RESULT = infinity.

INV_RESULT

For VAX data types, INV_RESULT = 0.

For IEEE data types, INV_RESULT = NaN.

invalid argument

See domain error.

max_float

The largest finite number representable in the floating-point data types. See Appendix A for more information on max_float values.

min_float

The smallest positive normalized nonzero number representable in the floating-point data types. See Appendix A for more information on min_float values.

NaN

A floating-point value that is said to be "not a number" and contains an indeterminate quantity.

overflow

An exception condition caused by passing a floating-point value that is larger than the highest valid floating-point value. See max_float for additional information.

range error

An exception condition that occurs when a mathematically valid argument results in a function value that exceeds the range of representable values for floating-point data types.

underflow

An exception condition caused by passing a floating-point value that is lower than the lowest valid floating-point value. See min_float for additional information.

Index

A

B

C

D

E

F

G

H

I

J

L

M